AF Brief

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant:	Christopher R. SHEEDY	§	Confirmation No.:	6417
Serial No.:	09/764,526	§	Group Art Unit:	2122
Filed:	01/17/2001	§	Examiner:	C. Chow
For:	Method And Apparatus For Versioning Statically Bound Files	§	Docket No.:	200301937-1

RESPONSE TO NOTIFICATION OF NON-COMPLIANCE

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: November 2, 2004

Sir:

In response to the Notification of Non-Compliance with 37 CFR 1.192(c) dated October 22, 2004, Appellant hereby submits the attached revised Appeal Brief in connection with the above-identified application.

Respectfully submitted,

Mark E. Scott
PTO Reg. No. 43,100
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANT

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to: Commissioner for Patents, Alexandria, VA 22313-1450. Date of Deposit: 11/02/2004

Typed Name: Christina L. Paz

Signature

ORIGINAL

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Appellant:	Christopher R. SHEEDY	§	Confirmation No.:	6417
Serial No.:	09/764,526	§	Group Art Unit:	2122
Filed:	01/17/2001	§	Examiner:	C. Chow
For:	Method And Apparatus For Versioning Statically Bound Files	§	Docket No.:	200301937-1

APPEAL BRIEF

Mail Stop Appeal Brief – Patents
Commissioner for Patents
PO Box 1450
Alexandria, VA 22313-1450

Date: November 2, 2004

Sir:

Appellant hereby submits this Appeal Brief in connection with the above-identified application. A Notice of Appeal was filed via facsimile on August 17, 2004.

I. REAL PARTY IN INTEREST

The real party in interest is the Hewlett-Packard Development Company, a Texas Limited Partnership, having its principal place of business in Houston, Texas, through its merger with Compaq Computer Corporation (CCC) which owned Compaq Information Technologies Group, L.P. (CITG). The assignment from the CCC to CITG was recorded on December 2, 2003, at Reel/Frame 014177/0428.

II. RELATED APPEALS AND INTERFERENCES

Appellant is unaware of any related appeals or interferences.

III. STATUS OF THE CLAIMS

Originally filed claims:	1-13.
Claim cancellations:	None.
Added claims:	14-17.
Presently pending claims:	1-17.

Presently appealed claims: 1-17.

IV. STATUS OF THE AMENDMENTS

Appellant presented amendments to claims 14-17 in an after-final Response filed July 2, 2004. The Advisory Action dated August 6, 2004 entered the amendments for purposes of appeal, and indicated that the amendments overcame the Section 101 rejections of the final Office action dated May 19, 2004.

V. SUMMARY OF THE INVENTION

Various embodiments of the invention are directed to versioning statically bound files. (Specification Title). One way to build "a software system is to compile...the files of one or more libraries directly into the final executable file for the system (otherwise known as statically binding the files that make up the final executable file)." (Specification Page 1, lines 22-25). Some of the exemplary embodiments may be a computer implemented method comprising creating a source code file containing a function whose name comprises version information of a library (Specification Page 5, lines 23 – Page 6, line 7), compiling the source code file to create an object file placed within the library (Specification Page 6. lines 7-12; Figures 2, 3 and 6), and building an executable program using at least the function from the library such that the version information is contained in the executable program through the presence of the function (Specification Page 6, lines 13-29; Figures 4 and 7).

VI. ISSUE(S)

Whether claims 1-13 are obvious over Boehm (U.S. Pat. No. 6,457,170) in view of Evans (U.S. Pat. No. 5,805,899) and in further view of Tanaka (U.S. Pat. No. 6,665,735).

Whether claims 14-17 are obvious over Evans and Tanaka.

VII. GROUPING OF CLAIMS

Appellant proposes the following claim groupings:

Claims 1-13 stand together.

Claims 14-17 stand together.

The groupings above are for purposes of this appeal only. The groupings should not be construed to mean the patentability of any of the claims may be determined in later actions (*e.g.*, actions before a court) based on the groupings. Rather, the presumption of 35 U.S.C. § 282 shall apply to each claim individually.

VIII. ARGUMENT

A. Claims 1-13

Claim 1 is representative of this grouping of claims. Claim 1 stands rejected as allegedly obvious over Boehm, Evans and Tanaka.

Boehm appears to be directed to "a software system build method and apparatus that supports multiple users in a software development environment." (Boehm Title). With regard to version information, Boehm puts version information in source and object file names, and also embeds character strings with version information in object files.

> **The present invention utilizes a source and object file naming convention and identification scheme** to facilitate file identification and searching the network caches and creating the links to cached source and object files. **Source files are cached under their filenames and version numbers. Object files are cached under their filenames**, along with a random string of characters, which allows multiple versions of the same basic object file to simultaneously exist in the cache. **Object files also contain an embedded character string that identifies all source files, along with their version numbers, that were used to generate the object file**.

(Boehm Col. 4, lines 28-38 (emphasis added)). Because of this teaching, the Office action dated May 19, 2004 admits "Boehm does not expressly disclose: *a version function whose name comprises at least one of version information and product information*." (Office action dated May 19, 2004, Page 5, second full paragraph (emphasis original)). In an attempt to fill this deficiency of Boehm, the Office action relies on Evans.

Evans appears to be directed to a "method and apparatus for internal versioning of objects using a mapfile." (Evans Title).

> In the described embodiment of the invention, the global symbols and version names associated with each version are defined in a "mapfile" generated by a human being. The mapfile is input to the

> link-editor at build time along with one, or more, relocatable (compiled) objects to create a versioned object.

(Evans Col. 2, lines 25-32 (emphasis added)). When the mapfile is present, the link editor creates "special sections" in the shared object to hold version information. (Evans Col. 8, lines 58-65).

> The special sections are shown in FIG. 5 and include a version definition section 506, a version symbol section 508, and an optional version dependency section 510. ...
>
> FIG. 6 shows a format of a version definition section 506 of version shared object 114. FIG 16 shows an example of a version definition section based on Tables 1-3.

(Evans Col. 8, line 58 – Col. 9, line 1). Thus, version information in Evan's shared object is not in the form of a function name, as asserted in the Office action in the rejection of claim 1; but rather, the version information appears to be independent information included within the shared object. The Office action later admits that Evans does not put version information in the function names:

> **Evans does not expressly disclose a function whose name comprises version information**, the version information thus contained in the executable program through the presence of the function.

(Office action dated May 19, 2004, Page 16, first full paragraph (emphasis added)). Finally, it is noted that the Evans system is used not in statically bound systems, but in dynamic linking systems. (Evans Col. 1, lines 8-9).

Tanaka appears to be directed a "method of changing a dynamic link library function efficiently and a computer system for executing the same." (Tanaka Title). In particular, Tanaka discloses adding preprocessing and/or post-processing to dynamic link libraries (DLLs) without changing the original DLL.,

> Specifically, **a preprocess or a postprocess can be added to a dynamic link library function and executed without changing the dynamic link library function**, by replacing the dynamic link library name described in the header portion of the program module and[. sic] a function name, which is to be referred to by the program module, with other names, linking and executing a dynamic link library, which includes a function whose name is the same as the replaced function name and has the same name as the replaced

> library name, instead of the original dynamic link library, executing the original dynamic link library function using the function having the same name as the replaced function name, and executing the preprocess or postprocess.

(Tanaka Col. 6, lines 20-32 (emphasis added)). Thus, Tanaka's concern is adding pre- and post-processing to dynamic link library functions, not getting version information to the final executable file. The Examiner relies on Tanaka Col. 6, lines 64-66 for an alleged teaching of "naming a function solely for the purpose of conveying information from a library." However, the cited section is not concerned with getting arbitrary information from the library, but instead is concerned specifically with adding the pre- and post-processing without altering the actual program module. (*See* Tanaka Col. 6, line 33 – Col. 7, line 30).

Claim 1, by contrast, specifically recites, "creating a version source file, the version source file containing **a version function whose name comprises at least one of version information and product information of the library**... ." Neither Boehm, Evans, nor Tanaka, alone or in combination, teach or fairly suggest a version function whose name comprises at least one of version information and product information of the library. The Office action dated May 19, 2004 admits as a matter of law that Boehm falls short of such a teaching. Evans teaches a version definition section 506, a version symbol section 508, and an optional version dependency section 510, but falls woefully short of teaching that version information should be a function name. Tanaka is likewise deficient. Thus, even if considered together (which Appellant does not admit is proper), Boehn, Evans and Tanaka fail to teach version information in the form of a function name. For this reason alone the rejections of claims 1-13 should be reversed, and the claims set for issue.

Claim 1 further specifically recites, "**building the executable program such that the version function whose name comprises at least one of version information and product information of the library is combined into the executable program**." Both Evans and Tanaka are expressly directed to dynamic linking (linking at runtime), and thus while there could be version

information in the executable program to help select the proper DLL, there may be no need to include the function in the executable program as it could be executed from the DLL. (*See, e.g.,* Tanaka Col. 7, lines 31-36 (noting that the neither the program module nor the module of the DLL are altered to accomplish the pre- and/or post-processing); Evans Col. 5, lines 18-31 ("runtime-linker 126 verifies that all versions of the shared object 114 needed by the dynamic executable are present. ... Runtime-linker 126 maps the objects in its memory and binds them together. ... Runtime-linker 126 performs a verification check to insure that the version of the shared object 114 required by the dynamic executable is present.")). For these additional reason, the rejections of claims 1-13 should be reversed, and the claims set for issue.

Based on the forgoing, Appellant respectfully submits that the rejections of the claims in this first grouping be reversed, and the claims set for issue.

B. Claims 14-17

Claim 14 is representative of this group of claims. Claim 14 stands rejected as allegedly obvious over Evans in view of Tanaka.

Evans appears to be directed to a "method and apparatus for internal versioning of objects using a mapfile." (Evans Title).

> In the described embodiment of the invention, the global symbols and version names associated with each version are defined in a "mapfile" generated by a human being. The mapfile is input to the link-editor at build time along with one, or more, relocatable (compiled) objects to create a versioned object.

(Evans Col. 2, lines 25-32 (emphasis added)). When the mapfile is present, the link editor creates "special sections" in the shared object to hold version information. (Evans Col. 8, lines 58-65).

> The special sections are shown in FIG. 5 and include a version definition section 506, a version symbol section 508, and an optional version dependency section 510. ...
>
> FIG. 6 shows a format of a version definition section 506 of version shared object 114. FIG 16 shows an example of a version definition section based on Tables 1-3.

(Evans Col. 8, line 58 – Col. 9, line 1). Thus, version information in Evan's shared object is not in the form of a function name; but rather, the version information appears to be independent information included within the shared object. The Office action admits that Evans does not put version information in the function names:

> **Evans does not expressly disclose a function whose name comprises version information**, the version information thus contained in the executable program through the presence of the function.

(Office action dated May 19, 2004, Page 16, first full paragraph (emphasis added)). Finally, it is noted that the Evans system is used not in statically bound systems, but in dynamic linking systems. (Evans Col. 1, lines 8-9).

Tanaka appears to be directed a "method of changing a dynamic link library function efficiently and a computer system for executing the same." (Tanaka Title). In particular, Tanaka discloses adding preprocessing and/or post-processing to dynamic link libraries (DLLs) without changing the original DLL file.

> Specifically, **a preprocess or a postprocess can be added to a dynamic link library function and executed without changing the dynamic link library function**, by replacing the dynamic link library name described in the header portion of the program module and[.] a function name, which is to be referred to by the program module, with other names, linking and executing a dynamic link library, which includes a function whose name is the same as the replaced function name and has the same name as the replaced library name, instead of the original dynamic link library, executing the original dynamic link library function using the function having the same name as the replaced function name, and executing the preprocess or postprocess.

(Tanaka Col. 6, lines 20-32 (emphasis added)). Thus, Tanaka's concern is adding pre- and post-processing to dynamic link library functions, not getting version information to the final executable file. The Examiner relies on Tanaka Col. 6, lines 64-66 for an alleged teaching of "naming a function solely for the purpose of conveying information from a library." However, the cited section is not concerned with getting arbitrary information from the library, but instead is

concerned specifically with adding the pre- and post-processing without altering the actual program module. (*See* Tanaka Col. 6, line 33 – Col. 7, line 30).

Claim 14, by contrast, specifically recites, "creating source code file containing a function whose name comprises version information of a library... ." The Office action dated May 19, 2004 admits that Evans fails to teach such a system. Thus, even if Evans and Tanaka are considered together (which Appellant does not admit is proper), the references together still fail to teach version information as a function name. For this reason alone the rejections of claims 14-17 should be reversed, and the claims set for issue.

Claim 14 further recites, "building an executable program using at least the function from the library such that the version information is contained in the executable program through the presence of the function." Both Evans and Tanaka are expressly directed to dynamic linking (linking at runtime), and thus while there could be version information in the executable program to help select the proper DLL, there may be no need to include the function in the executable program as it could be executed from the DLL. (*See, e.g.,* Tanaka Col. 7, lines 31-36 (noting that the neither the program module nor the module of the DLL are altered to accomplish the pre- and/or post-processing); Evans Col. 5, lines 18-31 ("runtime-linker 126 verifies that all versions of the shared object 114 needed by the dynamic executable are present. ... Runtime-linker 126 maps the objects in its memory and binds them together. ... Runtime-linker 126 performs a verification check to insure that the version of the shared object 114 required by the dynamic executable is present.")). For these additional reasons, the rejections of claims 14-17 should be reversed, and the claims set for issue.

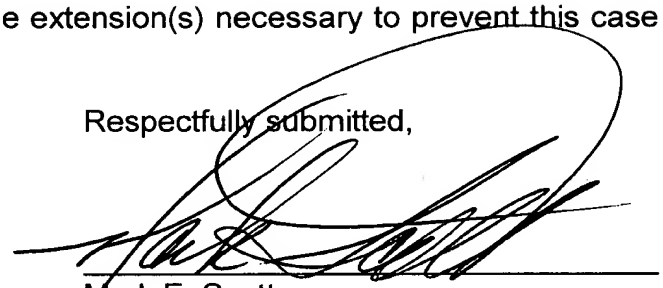
Based on the forgoing, Appellant respectfully submits that the rejections of the claims in this second grouping be reversed, and the grouping set for issue.

IX. CONCLUSION

For the reasons stated above, Appellant respectfully submits that the Examiner erred in rejecting all pending claims. If any fees or time extensions are inadvertently omitted or if any fees have been overpaid, please appropriately charge or credit those fees to Hewlett-Packard Company Deposit Account

Number 08-2025 and enter any time extension(s) necessary to prevent this case from being abandoned.

Respectfully submitted,



Mark E. Scott
PTO Reg. No. 43,100
CONLEY ROSE, P.C.
(713) 238-8000 (Phone)
(713) 238-8008 (Fax)
ATTORNEY FOR APPELLANT

HEWLETT-PACKARD COMPANY
Intellectual Property Administration
Legal Dept., M/S 35
P.O. Box 272400
Fort Collins, CO 80527-2400

APPENDIX TO APPEAL BRIEF

CURRENT CLAIMS

1. A computerized method of saving version and product information of a library in an executable program, comprising:

creating a version source file, the version source file containing a version function whose name comprises at least one of version information and product information of the library;

compiling the version source file to create a version object file;

rebuilding the library to include the version object file; and

building the executable program such that the version function whose name comprises at least one of version information and product information of the library is combined into the executable program.

2. A method of saving as recited in claim 1, wherein creating the version of the source file further comprises:

constructing a version string, the version string containing version and product identification information pertaining to the library;

combining function symbols with the version string to form the version function;

creating a version source file whose name includes a keyword and the name of the at least one library; and

storing the version function in the version source file.

3. A method of saving as recited in claim 1,

wherein creating the version source file further comprises:

constructing a version string containing version and product identification information pertaining to the library;

combining a build identifier and function symbols with the version string to form a name of the version function;

creating a version source file whose name includes a keyword and the name of the library; and

storing the version function in the version source file.

4. A method of saving as recited in claim 3, wherein the build identifier is a date on which the build occurs.

5. A method of saving as recited in claim 3, wherein the build identifier is a number that uniquely identifies the build.

6. A method of saving as recited in claim 3, wherein the build identifier of a user that performs the build.

7. A method of saving as recited in claim 1, wherein rebuilding the library further comprises:

removing any version object file from the library; and

remaking the library to include the version object file.

8. A method of saving as recited in claim 1, wherein building the executable includes:

creating a temporary storage area;

obtaining the version object file from the library, the version object file having a name that includes a keyword and the name of the library in which the version object file resides;

storing the version object file in the temporary storage area; and

compiling into the executable the stored version object file so that the executable contains the version and product information pertaining to the library.

9. A method of saving as recited in claim 1,

wherein there is a plurality of libraries; and

wherein creating, compiling and rebuilding are performed for each library of the plurality of libraries.

10. A method of saving as recited in claim 9,
further comprising, prior to the building step, selecting from the plurality of libraries a group of libraries needed for the building of the executable, each library in the group having a version object file; and
wherein building the executable includes:
creating a temporary storage area;
obtaining the version object file from each of the selected libraries, each version object file having a name that includes a keyword and the name of the library in which the version object file resides;
storing each of the version object files in the temporary storage area;
creating a list of the names of the stored version object files; and
compiling into the executable each of the stored version object files in the list so that the executable contains any functions needed by the executable from each library in the group and the version and product information of each library of the group.

11. A method of saving as recited in claim 9,
further comprising, prior to the building of the executable,
selecting a group of libraries from the plurality of libraries, each library in the group having a version object file;
building a compound library from the selected group of libraries, the compound library including a version object file for the compound library and the version object files of each library in the group; and
wherein building the executable includes building an executable to include the compound library, such that the version and product information

of the compound library and each library in the selected group are combined into the executable program.

12. A method of saving a recited in claim 11, wherein building a compound library further comprises:

creating a temporary storage area for holding the object files of each library of the selected group and the version object file for the compound library;

extracting all object files, including the version object files, from each library of the selected group;

storing the extracted object files for all libraries of the selected group in the temporary storage area;

creating a version source file for the compound library, the version source file for the compound library containing version and product information pertaining to the compound library;

compiling the version source file to create a version object file for the compound library;

storing the version object file in the temporary storage area;

building the compound library from the object files stored in the temporary storage area;

saving the compound library in a library storage area; and

deleting the temporary storage area.

13. A method of saving as recited in claim 9,

further comprising, prior to the building of the executable,

selecting a group of libraries from the plurality of libraries, the group including at least one compound library, and each library in the group having at least one version object file; and

building a multiple compound library from the selected group of libraries, the multiple compound library including a

version object file for the multiple compound library and the version object files of each library in the group; and

wherein building the executable includes building an executable to include the multiple compound library, such that the version and product information of the multiple compound library and each library in the selected group are combined into the executable program.

14. A computer implemented method comprising:

creating source code file containing a function whose name comprises version information of a library;

compiling the source code file to create an object file placed within the library; and

building an executable program using at least the function from the library such that the version information is contained in the executable program through the presence of the function.

15. The computer implemented method as defined in claim 14 further comprising creating the source code file containing a void function with an empty body.

16. The computer implemented method as defined in claim 14 further comprising creating the source code file containing a non-void function with text in the body.

17. The computer implemented method as defined in claim 14 further comprising creating the source code file containing the function whose name also comprises product information, and wherein the product information is contained in the executable program through the presence of the function.